Hertentamen Vertalerbouw—30 augustus 2004

De gecorrigeerde tentamens zijn af te halen op het onderwijsbureau.

Opmerkingen:

- Schrijf **netjes** en duidelijk, met zwarte of blauwe pen.
- Zet op het eerste blad alle gegevens als naam, etc., en het totaal aantal ingeleverde bladen, en nummer de ingeleverde bladen.
- Lees de opgaven eerst goed door.
- Motiveer uw antwoorden.
- De opgaven zullen gewogen meetellen in het totaalcijfer, volgens de vermelde bestedingstijd.

1. (40 minuten)

   Gegeven zijn de volgende producties:

   $$\begin{array}{cccl} & E & \rightarrow & FbG \\ , & F & \rightarrow & aE \\ , & F & \rightarrow & aG \\ , & G & \rightarrow & \\ , & G & \rightarrow & cE \end{array}$$

   a) Geef voor alle nonterminals uit deze produkties de sets *first* en *follow*.

   b) Is de grammatica, gegeven door de produkties (startsymbool is E): $LL(1), LR(0),\ SLR(1),\ LR(1)$?
   Geef in geval van conflicten deze duidelijk aan.

2. (50 minuten)

Gegeven is een eenvoudig taaltje, dat syntactisch gespecificeerd wordt door:

	B :	**Def** Ds | [E] |
,	Ds :	D ; Ds
,	Ds :	D
,	D :	id Ft equal E
,	Ft :	(Pl) : T
,	Pl :	P Ps
,	Ps :	, Pl
,	Ps :	
,	P :	T id
,	T :	**Int**
,	T :	**Real**
,	E :	id
,	E :	id (Al)
,	E :	intdenot
,	E :	realdenot
,	Al :	E Es
,	Es :	, Al
,	Es :	

Ziehier een (korrekt) voorbeeld uit dit taaltje:

```
Def
    p (Int i, Real r) : Real = 17.0
;   q (Real r) : Real = r
|[
    p(7,q(1.2))
]|
```

Het is de bedoeling deze syntaxregels te voorzien van attributen. De volgende restricties dienen opgelegd te worden:

- gebruik van niet-gedeclareerde identifiers wordt gesignaleerd;
- gebruik van dubbel-gedeclareerde identifiers wordt gesignaleerd;
- ongelijk aantal argumenten en parameters wordt gesignaleerd;
- ongelijk type van argument en parameter wordt gesignaleerd;

Geef bij elk grammatica symbool aan welke attributen erbij horen, en van ieder attribuut of het inherited danwel synthesized is. Declareer ook de door U gebruikte datastructuren en procedures.

U mag aannemen dat er een procedure `nextsym` is, die de terminal symbols (zoals de komma, id, intdenot, equal ...) herkent.

3. (45 minuten)

In een op Pascal lijkende programmeertaal met value en reference parameters is het volgende programma gegeven.

```
PROGRAM tentamen;

CONST upb = 10;

VAR a, b: integer;

PROCEDURE p1;
  VAR b, c: integer;
      d   : ARRAY [1..upb] OF integer;

  PROCEDURE p2 (x: integer);
    VAR a, b: integer;
    BEGIN ...
          b := x + c;                          (* 1 *)
          ...
    END (* p2 *);

  PROCEDURE q2 (VAR x: integer);
    VAR c, e: integer;
    BEGIN e := b;
          ...
          x := c + d[e];                       (* 2 *)
          ...
    END (* q2 *);

  BEGIN ...
        p2 (b);                                (* 3 *)
        ...
        q2 (d[b]);                             (* 4 *)
        ...
  END (* p1 *);

BEGIN ...  p1;  ...
END (* tentamen *).
```

Voor het geheugenbeheer en de adresberekeningen worden de volgende registers gebruikt:

`GP` het base address van het activation record van het hoofdprogramma,

`LNB` het base address van het huidige activation record, en

`LFA` het adres van de eerste vrije geheugenlokatie.

Voor het overdragen van de omgeving van een aan te roepen procedure kan het register `ENV` worden gebruikt.

In de machineinstructies `CALL` en `RETURN` van de doelmachine wordt impliciet gebruik gemaakt van een (aparte) return stack. U hoeft zich dus niet druk te maken over terugkeer-adressen!

Er zijn voldoende registers (R0, R1, R2, ...) voor het opslaan van de tussenresultaten.

a) Geef de layout van de activation records van $p1$, $p2$ en $q2$.

b) Geef de te genereren (pseudo-)instructies voor de procedure-entry en exit van $p1$ en $q2$.

c) Geef de te genereren (pseudo-)instructies voor de 4 gemarkeerde statements. Controle op index–waarden, die buiten array grenzen gaan, is niet nodig!

4. (45 minuten)

Zij G een uitgebreide contextvrije grammatica met de set P van produktieregels:

$$
\begin{array}{rlcll}
P = \{ & Z & \rightarrow & S\ eofs & \} \\
, & S & \rightarrow & Si & \\
, & S & \rightarrow & ifs\ E\ thens\ Si\ Ep\ fis & \\
, & E & \rightarrow & T\ Xt & \\
, & T & \rightarrow & ident & \\
, & T & \rightarrow & lpar\ E\ rpar & \\
, & Xt & \rightarrow & op\ T\ Xt & \\
, & Xt & \rightarrow & & \\
, & Ep & \rightarrow & elses\ S & \\
, & Ep & \rightarrow & & \\
, & Si & \rightarrow & assignment & \\
, & Si & \rightarrow & begs\ Sl\ ends & \\
, & Sl & \rightarrow & S\ St & \\
, & St & \rightarrow & sc\ S\ St & \\
, & St & \rightarrow & &
\end{array}
$$

(De terminal `sc` staat voor de semicolon ';'.)
Gegeven is: de grammatica G is $LL(1)$.

- Geef voor elke nonterminal in G een default productie aan.
- Geef de functiewaarden van de functie *follow* voor de nonterminals Xt en St.
- Toon aan dat de richters van de producties voor de nonterminal Xt disjunct zijn. Evenzo voor de nonterminal St.
- Geef de implementatie van het hoofdprogramma en van de procedures voor de nonterminals Si, Sl en St in een recursive descent parser voor G, *inclusief* syntactische error recovery.

Onderstaande declaraties mogen worden gebruikt in de implementatie van de recursive descent parser. Alle overige gebruikte procedures e.d. moeten gedeclareerd worden.

```
TYPE
  tsymbol    = (assignment, begs, elses, ends, fis, ident,
                ifs, lpar, op, rpar, sc, thens, eofs);
  tsymbolset = SET OF tsymbol;

VAR
  sym: tsymbol;

PROCEDURE initscanner;
  (* Initialisatie van de scanner *)
  BEGIN  ...  END (* initscanner *);

PROCEDURE nextsym;
  (* Levert bij aanroep de tokenwaarde op (in de variabele
     sym) van het eerstvolgende symbool in de invoer *)
  BEGIN  ...  END (* nextsym *);

PROCEDURE error (sy: tsymbol;  str: string);
  (* Genereert een foutmelding in de vorm:
     representatie van sy  waarde van str   *)
  BEGIN  ...  END (* error *);
```